A nice little scheduling problem

Yves Robert

ENS Lyon & Institut Universitaire de France
University of Tennessee Knoxville

yves.robert@ens-lyon.fr

CCDSC 2014 – Dareizé, September 3, 2014
A nice little scheduling problem

Lame motivation
A nice little scheduling problem

Theorem 1
Theorem 2
Theorem 3
Theorem 4
Theorem 5
Theorem 6
Theorem 8
Theorem 9
A nice little scheduling problem

Conclusion: proving Theorem 7 would be nice
Algorithms for coping with silent errors

Yves Robert

ENS Lyon & Institut Universitaire de France
University of Tennessee Knoxville

yves.robert@ens-lyon.fr

CCDSC 2014 – Dareizé, September 3, 2014
Exascale platforms

- **Hierarchical**
  - $10^5$ or $10^6$ nodes
  - Each node equipped with $10^4$ or $10^3$ cores

- **Failure-prone**

| MTBF – one node of $10^6$ nodes | MTBF – platform of $10^6$ nodes | 10 years 5mn | 120 years 1h |

More nodes $\Rightarrow$ Shorter MTBF (Mean Time Between Failures)
Definitions

- Instantaneous error detection $\Rightarrow$ fail-stop failures, e.g. resource crash
- Silent errors (data corruption) $\Rightarrow$ detection latency

Silent error detected only when the corrupt data is activated

- Includes some software faults, some hardware errors (soft errors in L1 cache), double bit flip
- Cannot always be corrected by ECC memory
Quotes

- **Soft Error**: An unintended change in the state of an electronic device that alters the information that it stores without destroying its functionality, e.g. a bit flip caused by a cosmic-ray-induced neutron. (Hengartner et al., 2008)

- **SDC** occurs when incorrect data is delivered by a computing system to the user without any error being logged (Cristian Constantinescu, AMD)

- **Silent errors are the black swan of errors** (Marc Snir)
Should we be afraid? (courtesy Al Geist)

Fear of the Unknown

**Hard errors** – permanent component failure either HW or SW (hung or crash)

**Transient errors** – a blip or short term failure of either HW or SW

**Silent errors** – undetected errors either hard or soft, due to lack of detectors for a component or inability to detect (transient effect too short). Real danger is that answer may be incorrect but the user wouldn’t know.

Statistically, silent error rates are increasing. Are they really? Its fear of the unknown

Are silent errors really a problem or just monsters under our bed?
Probability distributions for silent errors

Theorem: $\mu_p = \frac{\mu_{\text{ind}}}{p}$ for arbitrary distributions
Probability distributions for silent errors

**Theorem:** $\mu_p = \frac{\mu_{\text{ind}}}{p}$ for arbitrary distributions
Lesson learnt for fail-stop failures

(Not so) Secret data
- Tsubame 2: 962 failures during last 18 months so $\mu = 13$ hrs
- Blue Waters: 2-3 node failures per day
- Titan: a few failures per day
- Tianhe 2: wouldn’t say

$$T_{opt} = \sqrt{2\mu C} \quad \Rightarrow \quad \text{WASTE}_{opt} \approx \sqrt{\frac{2C}{\mu}}$$

Petascale: $C = 20$ min $\mu = 24$ hrs $\Rightarrow \text{WASTE}_{opt} = 17\%$

Scale by 10: $C = 20$ min $\mu = 2.4$ hrs $\Rightarrow \text{WASTE}_{opt} = 53\%$

Scale by 100: $C = 20$ min $\mu = 0.24$ hrs $\Rightarrow \text{WASTE}_{opt} = 100\%$
Lesson learnt for fail-stop failures

(Not so) Secret data

- Tsubame: 962 failures during last 18 months so \( \mu = 13 \text{ hrs} \)
- Blue Waters: 2-3 node failures per day
- Titan: a few failures per day
- Tianhe: wouldn’t say

Exascale \neq \text{Petascale} \times 1000

Need more reliable components
Need to checkpoint faster

Petascale:
\[ C = 20 \text{ min} \quad \mu = 24 \text{ hrs} \quad \Rightarrow \text{WASTE}_{\text{opt}} = 17\% \]

Scale by 10:
\[ C = 20 \text{ min} \quad \mu = 2.4 \text{ hrs} \quad \Rightarrow \text{WASTE}_{\text{opt}} = 53\% \]

Scale by 100:
\[ C = 20 \text{ min} \quad \mu = 0.24 \text{ hrs} \quad \Rightarrow \text{WASTE}_{\text{opt}} = 100\% \]
Lesson learnt for fail-stop failures

(Not so) Secret data
- Tsubame 2: 962 failures during last 18 months so $\mu = 13$ hrs
- Blue Waters: 2-3 node failures per day
- Titan: a few failures per day
- Tianhe 2: wouldn’t say

Silent errors:
detector latency $\Rightarrow$ additional problems

Petascale: $C = 20$ min $\mu = 24$ hrs $\Rightarrow WASTE_{opt} = 17\%$
Scale by 10: $C = 20$ min $\mu = 2.4$ hrs $\Rightarrow WASTE_{opt} = 53\%$
Scale by 100: $C = 20$ min $\mu = 0.24$ hrs $\Rightarrow WASTE_{opt} = 100\%$
Outline

1. General-purpose approach
2. Checkpointing and verification
3. Application-specific methods
Outline

1. General-purpose approach
2. Checkpointing and verification
3. Application-specific methods
General-purpose approach

- Last checkpoint may have saved an already corrupted state
- Saving $k$ checkpoints (Lu, Zheng and Chien):
  1. Critical failure when all live checkpoints are invalid
  2. Which checkpoint to roll back to?
General-purpose approach

- Last checkpoint may have saved an already corrupted state
- Saving $k$ checkpoints (Lu, Zheng and Chien):
  1. Critical failure when all live checkpoints are invalid
  2. Which checkpoint to roll back to?

Assume unlimited storage resources
Assume verification mechanism
Optimal period?

Error and detection latency

- $X_e$ inter arrival time between errors; mean time $\mu_e$
- $X_d$ error detection time; mean time $\mu_d$
- Assume $X_d$ and $X_e$ independent
Arbitrary distribution

\[ WASTE_{FF} = \frac{C}{T} \]

\[ WASTE_{Fail} = \frac{T}{2} + R + \frac{\mu_d}{\mu_e} \]

Only valid if \( \frac{T}{2} + R + \mu_d \ll \mu_e \)

**Theorem**

- Best period is \( T_{opt} \approx \sqrt{2\mu_e C} \)
- Independent of \( X_d \)
Limitation of the model

It is not clear how to detect when the error has occurred (hence to identify the last valid checkpoint)

Need a verification mechanism to check the correctness of the checkpoints. This has an additional cost!
Outline

1. General-purpose approach
2. Checkpointing and verification
3. Application-specific methods
Coupling checkpointing and verification

- Verification mechanism of cost $V$
- Silent errors detected only when verification is executed
- Approach agnostic of the nature of verification mechanism (checksum, error correcting code, coherence tests, etc)
- Fully general-purpose (application-specific information, if available, can always be used to decrease $V$)
Base pattern (and revisiting Young/Daly)

<table>
<thead>
<tr>
<th></th>
<th>Fail-stop (classical)</th>
<th>Silent errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pattern</td>
<td>$T = W + C$</td>
<td>$S = W + V + C$</td>
</tr>
<tr>
<td>$\text{WASTE}_{FF}$</td>
<td>$\frac{C}{T}$</td>
<td>$\frac{V+C}{S}$</td>
</tr>
<tr>
<td>$\text{WASTE}_{\text{fail}}$</td>
<td>$\frac{1}{\mu}(D + R + \frac{W}{2})$</td>
<td>$\frac{1}{\mu}(R + W + V)$</td>
</tr>
<tr>
<td>Optimal</td>
<td>$T_{\text{opt}} = \sqrt{2C\mu}$</td>
<td>$S_{\text{opt}} = \sqrt{(C + V)\mu}$</td>
</tr>
<tr>
<td>$\text{WASTE}_{\text{opt}}$</td>
<td>$\sqrt{\frac{2C}{\mu}}$</td>
<td>$2\sqrt{\frac{C+V}{\mu}}$</td>
</tr>
</tbody>
</table>

$\text{Waste} = 18/29$
With $p = 1$ checkpoint and $q = 3$ verifications

<table>
<thead>
<tr>
<th>Base Pattern</th>
<th>$p = 1, q = 1$</th>
<th>$WASTE_{opt} = 2 \sqrt{\frac{C+V}{\mu}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>New Pattern</td>
<td>$p = 1, q = 3$</td>
<td>$WASTE_{opt} = 2 \sqrt{\frac{4(C+3V)}{6\mu}}$</td>
</tr>
</tbody>
</table>
**Balanced Algorithm**

- \( p \) checkpoints and \( q \) verifications, \( p \leq q \)
- \( p = 2, \ q = 5, \ S = 2C + 5V + W \)
- \( W = 10w \), six chunks of size \( w \) or \( 2w \)
- May store invalid checkpoint (error during third chunk)
- After successful verification in fourth chunk, preceding checkpoint is valid
- Keep only two checkpoints in memory and avoid any fatal failure
**Balanced Algorithm**

1. \((\text{proba } 2w/W)\) \(T_{\text{lost}} = R + 2w + V\)
2. \((\text{proba } 2w/W)\) \(T_{\text{lost}} = R + 4w + 2V\)
3. \((\text{proba } w/W)\) \(T_{\text{lost}} = 2R + 6w + C + 4V\)
4. \((\text{proba } w/W)\) \(T_{\text{lost}} = R + w + 2V\)
5. \((\text{proba } 2w/W)\) \(T_{\text{lost}} = R + 3w + 2V\)
6. \((\text{proba } 2w/W)\) \(T_{\text{lost}} = R + 5w + 3V\)

\[WASTE_{\text{opt}} \approx 2\sqrt{\frac{7(2C + 5V)}{20\mu}}\]
Outline

1. General-purpose approach
2. Checkpointing and verification
3. Application-specific methods
ABFT: dense matrices / fail-stop, extended to sparse / silent. Limited to one error detection and/or correction in practice.

Asynchronous (chaotic) iterative methods (old work).

Partial differential equations: use lower-order scheme as verification mechanism (detection only, Benson, Schmit and Schreiber).

FT-GMRES: inner-outer iterations (Hoemmen and Heroux).

PCG: orthogonalization check every $k$ iterations, re-orthogonalization if problem detected (Sao and Vuduc).

... Many others.
On-line ABFT scheme for PCG

Zizhong Chen, PPoPP’13

- Iterate PCG
  - **Cost**: SpMV, preconditioner solve, 5 linear kernels
- Detect soft errors by checking orthogonality and residual
  - **Verification every** $d$ iterations
    - **Cost**: scalar product + SpMV
- Checkpoint every $c$ iterations
  - **Cost**: three vectors, or two vectors + SpMV at recovery
- Experimental method to choose $c$ and $d$
Conclusion

- Soft errors difficult to cope with, even for divisible workloads
- Investigate graphs of computational tasks
- Combine checkpointing and application-specific techniques
- Multi-criteria optimization problem
  - execution time/energy/reliability
  - best resource usage (performance trade-offs)

Several challenging algorithmic/scheduling problems 😊
A little game?

Framework

- Compute something
- Energy cost $C_1 = 10$ and failure probability $f_1 = 0.2$
- Energy cost $C_2 = 8$ and failure probability $f_2 = 0.3$
- ... (many other modes) ...

Problem

- You win when you get twice the same result (no false positive)
- Find optimal strategy and compute expected cost
A multicore game?

Framework

- Now you have $p$ cores for each trial
- Can freely run each core in a different mode (including idle)
- Each configuration has a cost $C$, and several probabilities:
  - $p_{tom} = \text{two or more successes (then you know you won)}$
  - $p_{one} = \text{exactly one success (but you don’t know it)}$
  - of course $f = 1 - p_{tom} - p_{one}$

Problem

- You win when you get twice the same result (no false positive)
- Find optimal strategy and compute expected cost
Back to task graphs?

Framework

- You’re given a (very big) task graph
- Each task produces files that you can save (checkpoint) or not
- Each task can choose from different execution speeds, with different error probabilities
- You can replicate some tasks, either for verification or for faster execution of successor tasks
- You may also be able to verify results by some application-specific mechanism

Problem

- Given energy budget or power cap, minimize execution time
- For each task, many things to be decided by schedule 😞
Back to task graphs?

Framework

- You’re given a (very big) task graph.
- Each task produces files that you can save (checkpoint) or not.
- Each task can be executed at different speeds, with different error probabilities.
- You can replicate some tasks, either for verification or for faster execution.
- You may also be able to verify results by some application-specific mechanism.

Problem

- Given energy budget or power cap, minimize execution time.
- For each task, many things to be decided by schedule.

Yves.Robert@ens-lyon.fr
Back to task graphs?

Framework

You’re given a (very big) task graph

Each task produces files that you can save (checkpoint) or not

Each task can choose from different execution speeds, with different error probabilities

You can replicate some tasks, either for verification or for faster execution of successor tasks

You may also be able to verify results by some application-specific mechanism

Problem

Given energy budget or power cap, minimize execution time

For each task, many things to be decided by schedule

The fox wants to save the polar bears . . .
Thanks

INRIA & ENS Lyon
- Anne Benoit
- Frédéric Vivien
- PhD students (Guillaume Aupy, Dounia Zaidouni)

Univ. Tennessee Knoxville
- George Bosilca
- Aurélien Bouteiller
- Jack Dongarra
- Thomas Hérault

Others
- Franck Cappello, Argonne National Lab.
- Henri Casanova, Univ. Hawai‘i
- Saurabh K. Raina, Jaypee IIT, Noida, India
- Marc Snir, Argonne National Lab.